DOI:
10.1007/978-3-319-59930-4_39

Document Version
Peer reviewed version

Demonstration: Multi-agent System for Distributed Cache Maintenance

Santhilata Kuppili Venkata[1], Katarzyna Musial[2], Samhar Mahmoud[1] and Jeroen Keppens[1]

[1] Department of Informatics, King's College London, London, UK
{santhilata.kuppili_venkata}@kcl.ac.uk
[2] Faculty of Science and Technology, Bournemouth University, Poole, UK

Abstract. Innovations in science and technology is increasing the demand on huge data transfers and hence number of data caches. In this paper, we consider the community caching solution, CommCache, where many groups of users are working together on related projects distributed all over the world. We demonstrate the use of proactive caches for data placement problem with the help of multi-agent coordination.

Keywords: Distributed cache, agent based modelling, coordination strategies

1 Introduction

Construction of models and simulations using multi-agent systems (MAS) is not new. Architectures using MAS enable to create applications such as distributed situation assessment, coordination etc. help researchers to develop new insights [2]. We utilise this property to represent distributed data caching. When groups of users working on similar projects access data from multiple databases, often they need the same data at different locations at different times. Also, their queries to the databases overlap significantly. Distributed caching is a complex system consists of components such as data servers, communication networks, middleware cache storage units, cache server (processing resources), and users. Traditionally, cache storage units are small in size. Hence during the cache maintenance process, a decision has to be made about storing in cache units the most relevant data and removing the obsolete data. This means that we have to identify **'what data'** to store, **'where'** a given data segment should be stored, and for **'how long'**. With the goal to reduce the response time and overall data transfers, multiple cache units need to coordinate together to cache each unit of data segment at an appropriate cache unit. Typical diagnostics used for decision making in placing data segments are: *frequency* of each data segment queried, *time* when a data segment was used, *location* preference where the data segment was requested, *association* among data segments at a given location, *number of joins* in a query, storage *capacity* of the cache unit, and *workload characteristics* depicting the pattern of query requests. We have designed **CommCache**, an agent based community cache framework to represent the distributed cache environment. In this paper, we examine five coordination strategies to represent centralised and peer-to-peer architectures.

2 Coordination Strategies Among Multiple Agents

The multi-agent model is shown in Fig 1. User agents (UA) are modelled as the software representation of humans that query databases. Query response time is measured as the time elapsed from the query sent from UA to the reply received by a user (Fig 1). Query analysis agent (QAA) assumes coordinator role in the distributed caching. It has combined responsibilities for analysis and management. Cache agents are designed to take active part in cache maintenance. They are cooperative agents. Cache agents handle local data during active phase and prepare meta data to be used during maintenance phase. Placement agent is an executor agent in the cache maintenance phase. It revises and recreates data placement plans and supports QAA. Database agents are resource (passive) agents. Other supporting agents are not discussed as they are not part of the demo. We choose the most common strategies used in distributed computing [1].

In **Master/slave coordination** strategy, query analysis agent (QAA) (usually a proxy server) acts as the master coordinating agent. With the help of a planning agent, master follows greedy strategy and ensures to place each data segment at a first available best position. **Voting strategy** enables cache agents to vote for the QAA's (coordinator) decisions. Cache agents participate pro-actively to vote based on the local knowledge (bias) such as affinity among all data stored within a cache unit. A plan is accepted when it is accepted by majority of voters. In **Multi-agent planning** strategy, cache agents develop individual plans keeping local benefits as heuristics. The Placement Agent acts as coordinator and resolves conflicts and develops a new global plan. All the above three are examples of centralised architecture. **Negotiation** allows peer to peer communication with other cache agents to discuss plans. Agents negotiate with each other till they reach to a mutually agreed solution. **Feedback strategy** employs a negotiation agent to provide feedback after every iteration to cache agents. When negotiations are not contributing to the improvement in the performance, negotiation agent may provide negative feedback refraining concerned agents from further negotiations.

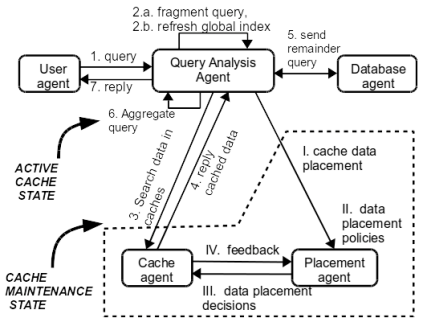

Fig. 1. Multi-agent architecture for distributed cache

3 Demonstration

Description of the Scenario: To evaluate the proposed multi-agent system for distributed cache maintenance, we narrate a fictitious scenario of television watching patterns. Many television viewers residing within nearby student accommodation blocks watch television programmes using the Internet streaming catchup television services. The catchup services allow to download television shows and watch up to a predefined

number of days. In order to reduce the volume of data transfers and hence the costs of Internet downloading, it was decided to install a cache storage unit at each of the accommodation block. Cache units are inter-connected with each other. For this scenario, we consider each user request to download programme(s) is considered as a query. A user can request a single programme (or shows) or multiple programmes within a single query. For the centralised architecture, a query analysis agent examines each request and generates sub-queries (partial queries) by each of the show requested in the query. This agent finds an appropriate place for the storage of each of the shows. In peer-to-peer architecture, cache storage units decide among themselves where to place the show. Patterns are obtained from the requests containing more than one show. The patterns thus obtained are used for the relocation of a show from one cache unit to another. Relocation clears the cache from storing shows that are locally not popular. A least frequently watched show is evicted as the cache refresh policy.

Experiment: We have generated workloads (list of 30,000 requests to watch shows over a period of seven days) using various statistical distributions for the evaluation (workload generation is not part of the demo). Viewers are given with a choice of 100 unique shows to choose from. About 35 shows are repeatedly requested according to a poisson distribution. Each show requires from 0.75GB to 2GB of memory space. All requests are made to watch minimum two shows or at the most three shows in a row only. Each request is identified by a unique identification number to determine the frequency (the popularity) of a particular show/ sequence of shows. We have set up a cache network with five cache storage units. There are 5 cache units set up for the execution of this experiment. We have developed a Java based simulator to test and evaluate the coordination strategies.

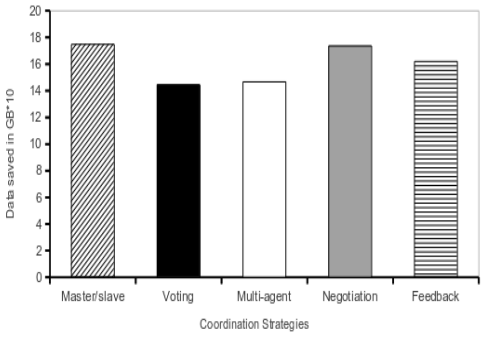

Fig. 2. Comparison of volume of data transfers saved

For the evaluation we need multiple metrics to be calculated based on the type of application. For example, in the above scenario, we need to compare the volume of data transfers saved as the result of coordination. The performance of different strategies for data placement for viewing requests is shown in the Fig 2. Voting and Multi-agent planning show considerable advantage over others in this case. A screen shot of the demo is shown in Fig 3. It accepts a configuration file to setup the distributed environment and the query input workload file in a predefined XML format[3]. System calculates the performance with respect to each of the metrics across strategies. The best strategy for the given input conditions will be implemented during maintenance. This demo demonstrates the decision making with the help of two bar charts: one (on the left), displays a comparison of strategies for the chosen performance metric. The second chart (on

[3] Generation of XML files is not part of this demo

the right) displays a comparison of the performance metric (volume of data transfers in this case) with the chosen coordination strategy (master-slave here) and no coordination among agents at all.

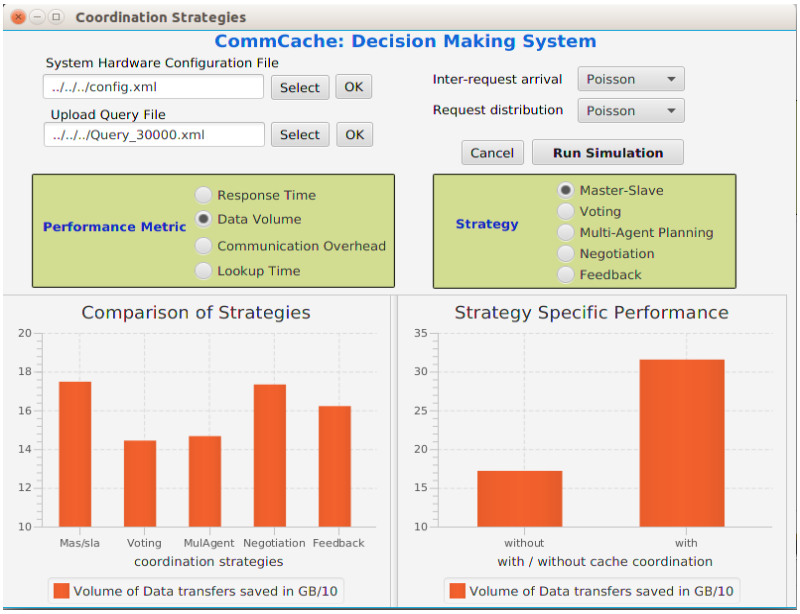

Fig. 3. A demonstration for volume of data transfers saved with Master-slave strategy

4 Conclusion

In this paper, we have described the applicability of multi-agent system for distributed data caching with the help of an example scenario. Implementation of coordination strategies are tested on the simulator for given query workloads. A demonstration is given with the help of five most suitable coordination strategies in the distributed environment. We would like to demonstrate comparison of more metrics in future.

References

1. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems - Concepts and Design*. Addison Wesley Publ. Comp., 5 edition, 2011.
2. K. Kravari and N. Bassiliades. A Survey of Agent Platforms . *Journal of Artificial Societies and Social Simulation*, 18, 2015.